Communications of the IIMA

Volume 6 | Issue 2 Article 8

2006

Feasible Task Schedules With Minimum Project Cost Solved By A Genetic Algorithm

Michael L. Gargano
Pace University

Louis V. Quintas
Pace University

Feasible Task Schedules With Minimum Project Cost Solved By A Genetic Algorithm

Michael L. Gargano
Computer Science Dept., Seidenberg School of Computer Science and Information Systems
Pace University, New York, NY 10038 mgargano@pace.edu

Louis V. Quintas
Mathematics Dept., Dyson College of Arts and Sciences
Pace University, New York, NY 10038 lquintas@pace.edu

ABSTRACT

Suppose that a project consists of n separate tasks and one and only one task can be completed in one time period. However, since some tasks can be started only before others have been completed, only feasible task schedules are considered. There is a cost associated with the time at which a task is completed and the project cost is equal to the sum of all the task costs. How can a feasible task schedule with minimum project cost be found for completing the entire project? This research proposes using a genetic algorithm to solve this problem efficiently.

INTRODUCTION

Consider a project that consists of n separate tasks that are to be completed over a sequence of time intervals such that exactly one task can be completed in any one time period. Further assume, some tasks can be started only before others have been completed, thus only feasible task schedules are to be considered. There is a cost associated with the time at which a task is completed and the project cost is equal to the sum of all the task costs. How can a feasible task schedule with minimum project cost be found for completing the entire project? This research proposes using a genetic algorithm (GA) to efficiently solve this problem.

Here is a simple example of a project with n = 7 tasks. The diagram below (Figure 1) represents the constraints for this project (see the comments that follow Figure 1).

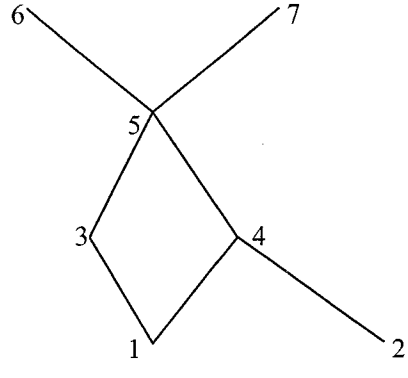

Figure 1: Diagram showing the constraints on the order in which tasks can be done.

In this diagram if two tasks have a line joining them then the task in the lower position of the diagram must be done before the other. For example, task 3 must be done before task 5 and 5 must be done before 6. Therefore, it can be implied that 3 must also be done before 6.

The matrix below (Figure 2) is another representation for this diagram. The entry in the i^{th} row and j^{th} column is 1 if task i is connected to task j. Since task j is in a higher position on the diagram, i must be completed before j; otherwise the entry is 0.

0	0	1	1	0	0	0
0	0	0	1	0	0	0
0	0	0	0	1	0	0
0	0	0	0	1	0	0
0	0	0	0	0	1	1
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 2: Matrix showing the constraints on the order in which tasks can be done.

The cost of completing task i at time t is given by the following table.

Time	**1**	**2**	**3**	**4**	**5**	**6**	**7**
task 1	2	3	3	3	4	4	5
task 2	2	4	2	8	8	6	6
task 3	1	1	2	2	3	3	3
task 4	2	1	1	3	4	5	6
task 5	5	5	4	4	4	3	3
task 6	4	4	4	4	4	4	4
task 7	2	3	2	3	2	3	2

Table 1: Cost of doing task i at time t.

There are ten possible feasible schedules that satisfy the constraints given in the diagram above. They are listed below with their associated costs calculated using the cost table above.

Feasible Schedule	Ordered List of Tasks	Project Cost
1	1 2 3 4 5 6 7	21
2	1 2 3 4 5 7 6	22
3	1 2 4 3 5 6 7	19
4	1 2 4 3 5 7 6	20
5	1 3 2 4 5 6 7	18
6	1 3 2 4 5 7 6	19
7	2 1 3 4 5 6 7	20
8	2 1 3 4 5 7 6	21
9	2 1 4 3 5 6 7	18
10	2 1 4 3 5 7 6	19

Table 2: All possible feasible schedules and the resulting project cost.

In this example there are two minimum valued solutions (i.e., feasible schedules 5 and 9). These two solutions both attain a minimum cost of 18.

MATHEMATICAL MODEL

Given a task constraint diagram (also called a Hasse diagram) or its equivalent matrix on the set of tasks $T = \{ t_1, t_2, t_3, \ldots, t_n \}$, a binary relation for T can be implied and this relation is a partial order (i.e., its reflexive, antisymmtric, and transitive). A feasible schedule is then just a total order on T that satisfies the constraints of the partial order. Given any finite set of tasks T and a partial order on T, there must be at least one task t (not necessarily unique) that can be completed in the first period (if there is more than one choice a random selection is made for t). Furthermore, removing this task from the diagram (or matrix) after its completion results in another partial order on the set $T - \{ t \}$. Thus, this process can be reiterated until all the tasks are completed and the project is finished. This process, which is also known as the topological sort algorithm, results in a feasible task schedule (i.e., a total ordering on T).

The relevant mathematical statements include:

Theorem 1. A partial order on a finite non-empty set T has a minimal element t and the relation restricted to T – { t } is also a partial order.

Topological Sort Algorithm

Input(T and a partial order on T)
p = 1
while T is non-empty
t_p = a minimal element from T (if there is a choice choose randomly)
T = T – { t_p }
p = p + 1
Output(a feasible task schedule $t_{i1}, t_{i2}, t_{i3}, \ldots, t_{in}$, that is, a permutation of $t_1, t_2, t_3, \ldots, t_n$ that is feasible)

GENETIC ALGORITHM METHODOLOGY

A *genetic algorithm* (GA) is a biologically inspired, highly robust heuristic search procedure that can be used to find optimal (or near optimal) solutions to intractable problems. The GA paradigm uses an adaptive methodology based on the ideas of Darwinian natural selection and genetic inheritance on a population of potential solutions. It employs the techniques of crossover (or mating), mutation, and survival of the fittest to generate new, typically fitter members of a population over a number of generations.

We propose a GA for efficiently solving this problem. Our GA creates and evolves a population of potential solutions (i.e., feasible task schedules) so as to facilitate the creation of new feasible members of the population by standard mating and mutation operations. A feasible search space contains only members that satisfy the problem constraints. By making use of problem-specific representations (i.e., a topological sort on T), our genetic algorithm insures a feasible search space during the classical operations of crossover and mutation and, in addition, eliminates the need to screen during the generation of the initial population. We adapted many of the standard GA techniques to this problem. A brief description of these techniques follows. Selection of parents for mating involves randomly choosing one very fit member of the population (i.e., one with a small project cost) and the other member randomly. The reproductive process is a simple crossover operation whereby two randomly selected parents are cut into sections at some randomly chosen position and then have the parts of their task schedules swapped to create new offspring (children). Mutation is performed by randomly choosing a member of the population, cloning it, and then modifying it to a slightly different task schedule. A grim reaper mechanism replaces low scoring members in the population with newly created more fit offspring and mutants. Our fitness measure will be the project cost, that is, the sum of the task costs. The GA is terminated when, either no improvement in the best fitness value is observed after pre-specified number of generations have been examined, and/or a satisficing solution is attained (i.e., the result is not necessarily optimum, but is satisfactory - for example, if the project cost is within budget limits).

Representing Members of the Population

In this example the population of potential solutions are simply feasible task schedules (i.e., ordered lists of tasks that satisfy the diagram below). Two potential solutions are

1 2 3 4 5 6 7 8 9 10 11 12 and 3 1 7 2 4 6 5 8 10 9 12 11

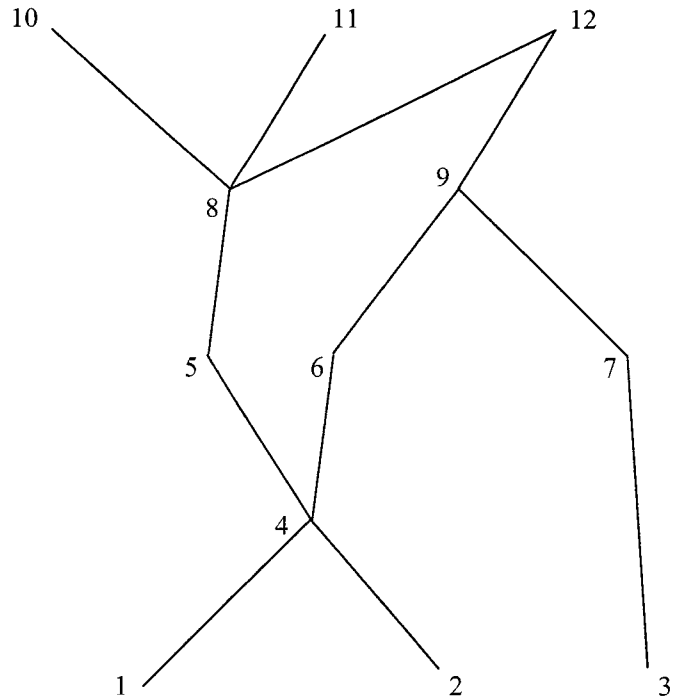

Figure 3: Diagram showing the constraints on the order in which tasks can be done.

Initial Population

The GA population will consist of N randomly generated feasible task schedules. The initial population (i.e., generation 0) is created by executing the topological sort algorithm N times. Since the topological sort algorithm will randomly choose the next minimal element, many different feasible task schedules will be created.

Mating and Mutating

Selecting parents for mating involves randomly choosing one of the fitter members of the population (i.e., one with a small project cost) and the other member randomly. The reproductive process is a simple crossover operation whereby two randomly selected parents are cut into two sections at some randomly chosen position k and then, as described in the following theorem, have the parts of their task schedules swapped to create two new offspring (children).

Theorem 2. If $P_1 = t_{i1}, t_{i2}, t_{i3}, t_{i4},\ldots t_{i\,k}, t_{i\,k+1},\ldots, t_{i\,n-2}, t_{i\,n-1}, t_{i\,n}$
and $P_2 = t_{j1}, t_{j2}, t_{j3}, t_{j4},\ldots t_{j\,k}, t_{j\,k+1},\ldots, t_{j\,n-2}, t_{j\,n-1}, t_{j\,n}$
are partial orders on a finite non-empty set T (i.e., the parents), then the children (offspring)
$C_1 = t_{i1}, t_{i2}, t_{i3}, t_{i4},\ldots t_{i\,k}, t_{j\,k+1*},\ldots, t_{j\,n-2*}, t_{j\,n-1*}, t_{j\,n*}$
with $\{ t_{j\,k+1*},\ldots, t_{j\,n-2*}, t_{j\,n-1*}, t_{j\,n*} \} = T - \{ t_{i1}, t_{i2}, t_{i3}, t_{i4},\ldots t_{i\,k} \}$ and
$t_{j\,k+1*},\ldots, t_{j\,n-2*}, t_{j\,n-1*}, t_{j\,n*}$ having its ordering the same as in P_2
and $C_2 = t_{i1*}, t_{i2*}, t_{i3*}, t_{i4*},\ldots t_{i\,k*}, t_{j\,k+1},\ldots, t_{j\,n-2}, t_{j\,n-1}, t_{j\,n}$
with $\{ t_{i1*}, t_{i2*}, t_{i3*}, t_{i4*},\ldots t_{i\,k*} \} = T - \{ t_{j\,k+1},\ldots, t_{j\,n-2}, t_{j\,n-1}, t_{j\,n} \}$ and
$t_{i1*}, t_{i2*}, t_{i3*}, t_{i4*},\ldots t_{i\,k*}$ having its ordering the same as in P_1

are also partial orders on the finite non-empty set T.

Here is an illustration of mating based on Figure 3.

If k = 6 and P_1 = 1 2 3 4 5 6 7 8 9 10 11 12 and
P_2 = 3 1 7 2 4 6 5 8 10 9 12 11

then C_1 = 1 2 3 4 5 6 7 8 10 9 12 11
C_2 = 1 2 3 4 6 7 5 8 10 9 12 11.

Notice that the first 6 = k positions of C_1 are the same as P_1 while the last 6 = n – k reflect the order of 7 8 9 10 11 12 as seen in P_2, that is, 7 8 10 9 12 11 and similarly note that the last 6 = n – k positions of C_2 are the same as P_2 while the first 6 = k reflect the order of 3 1 7 2 4 6 as seen in P_1, that is, 1 2 3 4 6 7.

In a similar manner an illustration of mutation based on Figure 3 can be shown. Starting with P = 2 1 4 5 6 8 11 3 7 10 9 12 then M = 2 1 4 6 5 8 11 3 7 10 9 12 with k = 7 is obtained by cloning the last 5 = n – k positions of P and executing a topological sort on the partial order restricted to the tasks numbered 2 1 4 5 6 8 11 (e.g., one such topological sort may be 2 1 4 6 5 8 11) and finally placing the result in the first k = 7 positions of M.

Fitness

There is a cost associated with the time at which each task is completed (i.e., C (task, time)) defined by a cost table. The fitness associated with a potential solution (i.e., ordered list of tasks (feasible schedule)) is simply the sum of the costs of completing each task in its time period.

If there are n tasks $t_1, t_2, t_3, \ldots, t_n$ and if $t_{i1}, t_{i2}, t_{i3}, \ldots, t_{in}$ is a feasible schedule (i.e., a permutation of the tasks that forms a feasible schedule) then task t_{ij} is completed in time period j with associated cost C (t_{ij}, j). The fitness of this feasible task schedule is given by

$$\text{Fitness}(t_{i1}, t_{i2}, t_{i3}, \ldots, t_{in}) = \Sigma\, C(t_{ij}, j) \text{ summed over all the tasks} = \text{project cost.}$$

The Grim Reaper

After creating a number of children and mutants, the population will grow from its initial size N to N + E. The grim reaper reduces the enlarged population back down to N by eliminating the population members with poorest fitness.

The Genetic Algorithm

We now state the genetic algorithm we used:

1) Randomly initialize a population of potential solutions.
2) Calculate the fitness of any population member not yet evaluated.
3) Sort the members of the population in order of fitness.
4) Randomly select parents for mating, generate offspring using crossover at random positions, and evaluate offspring.
5) Randomly select and clone members of the population, generate mutations by changing them in random positions, and evaluate mutants.
6) Sort all the members of the expanded population in order of fitness.
7) Use the grim reaper to eliminate the population members with poor fitness.
8) If (termination criteria is met) then return best population member(s)

ELSE go to step 4.

CONCLUDING REMARKS

For an arbitrary project with a large number of tasks finding a minimum cost task schedule is a difficult problem and solving it by brute force methods is computationally prohibitive. Thus, it is proposed that using a genetic algorithm approach to solve this problem can be a very effective method for obtaining optimal/near optimal solutions efficiently.

Acknowledgements

We wish to thank Pace University's Seidenberg School of Computer Science and Information Systems for partially supporting this research.

REFERENCES

Davis, L. (1991) Handbook of Genetic Algorithms, Van Nostrand Reinhold.

Edelson, W. and M.L. Gargano (1998) Minimal Edge-Ordered Spanning Trees Solved By a Genetic Algorithm with Feasible Search Space, *Congressus Numerantium* 135, 37-45.

Gargano, M.L., W. Edelson, (2001) Optimally Sequenced Matroid Bases Solved By A Genetic Algorithm with Feasible Search Space Including a Variety of Applications, *Congressus Numerantium* 150, 5-14.

Gargano, M.L., Maheswara Prasad Kasinadhuni, (2004) Self-adaption in Genetic Algorithms using Multiple Genomic Redundant Representations, *Congressus Numerantium* 167, 183-192.

Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley.

Mitchell, M. (2001) An Introduction to Genetic Algorithms, MIT Press.

Rosen, K.H. (1998) Discrete Mathematics and Its Applications, 4th Ed, Random House.